Assignment 3

Spring '24 - Anushree Kolhe

In [1]:
```
import networkx as nx
```

Section 1: Friendships Network

In [2]:
```
G = nx.read_graphml('Data/highschool_2012.graphml')
print(nx.info(G))

total_nodes = nx.number_of_nodes(G)
total_edges = nx.number_of_edges(G)
```

```
Graph with 180 nodes and 2220 edges

/var/folders/_3/98b473cn1k993m8ln5gwyv6h0000gn/T/ipykernel_58026/2221535143.py:2: DeprecationWarning: info is de
precated and will be removed in version 3.0.

  print(nx.info(G))
```

In [3]:
```
highest_degree = max(G.degree(), key=lambda x: x[1])
print(f'Node with highest degree: {highest_degree[0]}')
print(f'Highest degree: {highest_degree[1]}\n')
```

```
Node with highest degree: 826
Highest degree: 56
```

In [4]:
```
largest_cl_coeff = max(nx.clustering(G).items(), key=lambda x: x[1])
print(f'Node with largest clustering coefficient: {largest_cl_coeff[0]}')
print(f'Largest clustering coefficient: {largest_cl_coeff[1]}\n')
```

```
Node with largest clustering coefficient: 647
Largest clustering coefficient: 1.0
```

In [5]:
```
average_cl_coeff = round(nx.average_clustering(G), 4)
print(f'Average clustering coefficient of the n/w: {average_cl_coeff}\n')
```

```
Average clustering coefficient of the n/w: 0.4752
```

In [6]:
```
total_Mnodes = 0
total_Fnodes = 0

for node, attributes in G.nodes(data=True):
    if attributes['gender'] == 'M' :
        total_Mnodes += 1
    else :
        total_Fnodes += 1

proportion_male = round(total_Mnodes/total_nodes, 4)
proportion_female = round(total_Fnodes/total_nodes, 4)
```

In [7]:
```
print(f'Proportion of the nodes in the graph are male: {proportion_male}')
print(f'Proportion of the nodes in the graph are female: {proportion_female}\n')
```

```
Proportion of the nodes in the graph are male: 0.7333
Proportion of the nodes in the graph are female: 0.2667
```

In [8]:
```python
# Expected / Calculated values
expected_edges_ = {'M-M': 0, 'F-F': 0, 'M-F': 0}

expected_edges_['M-M'] = round(proportion_male*proportion_male*total_edges) #m^2
expected_edges_['F-F'] = round(proportion_female*proportion_female*total_edges) #f^2
expected_edges_['M-F'] = round(2*proportion_male*proportion_female*total_edges) #2mf


# Actual values
edges_ = {'M-M': 0, 'F-F': 0, 'M-F': 0}

for node1, node2, attr in G.edges(data=True):
    if G.nodes[node1]['gender'] != G.nodes[node2]['gender']:
        edges_['M-F'] += 1
    elif G.nodes[node1]['gender'] == 'M':
        edges_['M-M'] += 1
    else:
        edges_['F-F'] += 1
```

In [9]:
```python
print(f'Expected edge values:\n{expected_edges_}\n')
print(f'Actual edge values:\n{edges_}\n')
print(f'Expected sum: {sum(expected_edges_.values())}')
print(f'Actual sum: {sum(edges_.values())}\n')
```

```
Expected edge values:
{'M-M': 1194, 'F-F': 158, 'M-F': 868}

Actual edge values:
{'M-M': 1276, 'F-F': 182, 'M-F': 762}

Expected sum: 2220
Actual sum: 2220
```

In [10]:
```python
print(f'Expected edges M-F: {expected_edges_["M-F"]}\n')
print(f'Actual edges M-F: {edges_["M-F"]}\n')
```

```
Expected edges M-F: 868

Actual edges M-F: 762
```

Section 2: Club Membership Network

In [11]:
```python
G = nx.read_edgelist('Data/club_membership.edgelist', create_using=nx.Graph())
nx.info(G)
```

```
/var/folders/_3/98b473cn1k993m8ln5gwyv6h0000gn/T/ipykernel_58026/3996694401.py:2: DeprecationWarning: info is de
precated and will be removed in version 3.0.

  nx.info(G)
```

Out[11]: 'Graph with 40 nodes and 95 edges'

In [12]:
```python
total_nodes = nx.number_of_nodes(G)
total_edges = nx.number_of_edges(G)
```

What is the mean number of organizational affiliations per person in the data set?
What is the mean number of members per organization?

In [13]:
```python
people = []
organizations = []

for node in G.nodes():
    if node[0] == 'o':
        organizations.append(node)
    else:
        people.append(node)
```

In [14]:
```python
sum_people = 0
for person in people :
    sum_people += len(set(G.neighbors(person)))

mean_people = sum_people/len(people)
```

In [15]:

```
sum_org = 0
for org in organizations :
    sum_org += len(set(G.neighbors(org)))

mean_org = sum_org/len(organizations)
```

Function to measure similarity

In [16]:

```
def similarity_measure(G, node_1, node_2):

    if (node_1 in people and node_2 in people) or (node_1 in organizations and node_2 in organizations):
        neighbors_n1 = set(G.neighbors(node_1))
        neighbors_n2 = set(G.neighbors(node_2))
        sim = len(neighbors_n1.intersection(neighbors_n2))/len(neighbors_n1.union(neighbors_n2))
        return sim

    else:
        raise ValueError("Bipartite Graph!\n\tThe nodes need to be from the same group.")
```

In [17]:

```
# Will throw an error as the groups are different.
similarity_measure(G, 'o11', 'p13')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [17], in <cell line: 2>()
      1 # Will throw an error as the groups are different.
----> 2 similarity_measure(G, 'o11', 'p13')

Input In [16], in similarity_measure(G, node_1, node_2)
      7     return sim
      9 else:
---> 10     raise ValueError("Bipartite Graph!\n\tThe nodes need to be from the same group.")

ValueError: Bipartite Graph!
	The nodes need to be from the same group.
```

In [18]:

```
similarity_measure(G, 'o11', 'o6')
```

Out[18]:

```
0.3
```

Function to get find the pair of nodes with highest similarity value

In [19]:

```
def get_highest_similarity(G, grouptype):

    if grouptype == 'o': group = organizations
    else: group = people

    max_sim = 0
    pairs = []
    for i in range(len(group)-1):
        sim = similarity_measure(G, group[i], group[i+1])
        if sim > max_sim :
            max_sim = sim
            pairs = [group[i], group[i+1]]

    return(pairs, max_sim)
```

In [20]:

```
get_highest_similarity(G, 'o')
```

Out[20]:

```
(['o6', 'o15'], 0.3)
```

In [21]:

```
get_highest_similarity(G, 'p')
```

Out[21]:

```
(['p22', 'p3'], 0.75)
```

In [ ]:

```

```